Borst Automation Embedded Solutions
Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon: +49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/


# Real Time Serial SCM

## Overview

If real-time behavior is required when implementing a serial interface under Windows, then using the event system in Windows is out of the question. The only way to determine if the line is clear after a byte array has been sent is to wait the appropriate amount of time it takes.

In order to have a working source code that realizes this behavior, the *CSerial* class was provided. In this version, it was presented as a static class in order to initially simplify handling it.

The graphic below explains what is involved.

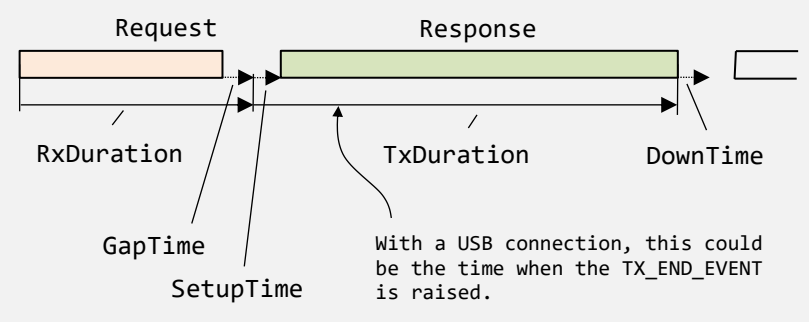


**Figure 1: Timing of a half-dublex connection (e.g. HART)**

The graphic shows that the time for receiving an array of bytes is made up of the actual transmission time and a gap time that is required to recognize the end of the byte array. When working with a carrier, the gap time is also needed to recognize that the carrier has been switched off.

The same applies to sending. Here the setup time is required to switch on the carrier.

## Parallel Processing

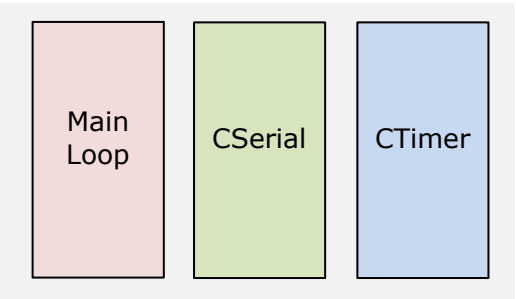


**Figure 2: Parallel processing**

In the implementation, three processes are executed in parallel. The Main Loop is running in the test client's app (exe). CSerial contains the code that performs the serial communication. Finally, CTimer is a class that, among other things, also implements a stopwatch with which the necessary times are recorded.

Walter Borst
Kapitaen-Alexander-Strasse 39
27472 Cuxhaven, GERMANY
Fon: +49 (0) 4721 6985100
E-Mail: walter.borst@borst-automation.de
Home: https://www.borst-automation.de/

# Development Environment

| | |
|---|---|
| IDE | Visual Studio 2019 (v142) |
| Language | ISO C++14 Standard |
| Windows SDK Version | 10.0 |

# Interface (CSerial)

| Declaration | Description | |
|---|---|---|
| **Initialization/Termination** | | |
| `void Init (uint8_t t_com_port, uint32_t t_baudrate)` | Possible com port numbers are 0..255.<br>Possible baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.<br>Init may only be called immediately after starting or after executing terminate. | |
| `void Terminate()` | Terminates all internal processes (threads) and eliminates all reserved resources. Terminate must be called before the application (exe) terminates. | |
| **Operation** | | |
| `TY_Status GetStatus()` | Gets the status of the serial communication. | |
| | NOT_SET(0) | Com port could not be initialized. Serial communication is not possible. The module does not respond to any call other than Terminate(). |
| | IDLE(1) | This state can be set from the IDLE, RECEIVE_DONE and TX_DONE states and results in switching off the receiver. |
| | READY(2) | The module is ready to receive data. |
| | RECEIVING(3) | At least 1 character was received. |
| | RECEIVE_DONE(4) | Bytes from reception are ready to be fetched. This state can only be changed by the application (main loop). This is done by using the EnableReceive() function to start the next reception of a byte stream or by PutTxBytes() to send kind of a response or DisableReceive() to enter the IDLE state. |
| | TX_RUNNING(5) | As soon as the method PutTxBytes() is called, this status is reported. After the last bit of the last character has been sent and the down time being passed (see figure 1). |
| | TX_DONE(6) | Three actions can take place in this state. Calling PutTxBytes causes another set of bytes to be sent while EnableReceive() switches back to the receive mode and DisableReceive() causes the status IDLE. |
| | TERMINATING(7) | After calling Terminate(), the software goes into this state until all tasks have been completed to the point that no 'garbage' remains. |
| | TERMINATED(8) | In the TERMINATED state, the only call that is executable is Init(). |
| `uint16_t GetRxBytes (uint8_t** tpp_bytes)` | This method only works in the RECEIVE_DONE state. The module returns a pointer to the bytes received. This pointer is valid until one of the methods PutTxBytes(), EnableReceive() or DisableReceive() is called. The direct return value indicates the number of characters received.<br>If this method is called in any other state, it returns a NULL pointer. Also the directly returned length is 0. | |
| `void PutTxBytes (uint8_t * tp_bytes, uint16_t t_size)` | This function can be called in the IDLE, RECEIVE_DONE and TX_DONE states and results in an array of bytes being sent. | |
| `void EnableReceive()` | Switches to the status READY. | |
| `void DisableReceive()` | Switches to the status IDLE. | |

# Files

| Name | Description |
|---|---|
| RealTimeSerialSCM.sln/vcxproj | Solution and project |
| MainLoop.cpp/h | The main loop |
| CFramework.cpp/h | Various global functions, macros and definitions |
| CSerial.cpp/h | Serial communications |
| CTimer.cpp/h | Timing issues |